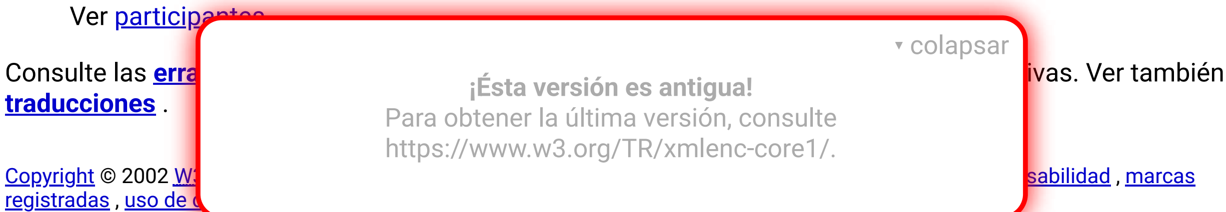


Sintaxis y procesamiento de cifrado XML

Recomendación del W3C del 10 de diciembre de 2002

Esta versión:
http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
Ultima versión:
http://www.w3.org/TR/xmlenc-core/
Versión previa:
http://www.w3.org/TR/2002/PR-xmlenc-core-20021003/
Editores
Donald Eastlake <dee3@torque.pothole.com>
José Reagle <reagle@w3.org>
Autores
Takeshi Imamura <IMAMU@jp.ibm.com>
Blair Dillaway <blaird@microsoft.com>
Ed Simón <edsimon@xmlsec.com>
Colaboradores
Ver participa



Consulte las **erra** ivas. Ver también **traducciones** .

Abstracto

Este documento especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden ser datos arbitrarios (incluido un documento XML), un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

Estado de este documento

Este documento es la Recomendación de cifrado XML (REC) del W3C . Este documento ha sido revisado por miembros del W3C y otras partes interesadas y ha sido respaldado por el Director como Recomendación del W3C. Es un documento estable y puede usarse como material de referencia o citarse como referencia normativa de otro documento. El papel del W3C al elaborar la Recomendación es llamar la atención sobre la especificación y promover su implementación generalizada. Esto mejora la funcionalidad y la interoperabilidad de la Web.

Esta especificación fue producida por el Grupo de Trabajo de Cifrado XML del W3C (Actividad), que cree que la especificación es suficiente para la creación de implementaciones interoperables independientes como se demuestra en el Informe de Interoperabilidad.

Las divulgaciones de patentes relevantes para esta especificación se pueden encontrar en la página de divulgación de patentes del Grupo de Trabajo de conformidad con la política del W3C.

Informe los errores en este documento a xml-encryption@w3.org (archivo público).

La lista de errores conocidos en esta especificación está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-errata .

La versión en inglés de esta especificación es la única versión normativa. La información sobre las traducciones de este documento (si corresponde) está disponible en http://www.w3.org/Encryption/2002/12-xmlenc-translations .

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of Contents

---

1 Introduction

This document specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption `EncryptedData` element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

When encrypting an XML element or element content the `EncryptedData` element replaces the element or content (respectively) in the encrypted version of the XML document.

When encrypting arbitrary data (including entire XML documents), the `EncryptedData` element may become the root of a new XML document or become a child element in an application-chosen XML document.

1.1 Editorial and Conformance Conventions

This specification uses XML schemas [XML-schema] to describe the content model.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC2119 [KEYWORDS]:

> "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmissions)"

Consequently, we use these capitalized keywords to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. These key words are not used (capitalized) to describe XML grammar; schema definitions unambiguously describe such requirements and we wish to reserve the prominence of these terms for the natural language descriptions of protocols and features. For instance, an XML attribute might be described as being "optional." Compliance with the XML-namespace specification [XML-NS] is described as "REQUIRED."

1.2 Design Philosophy

The design philosophy and requirements of this specification (including the limitations related to instance validity) are addressed in the XML Encryption Requirements [EncReq].

1.3 Versions, Namespaces, URIs, and Identifiers

No provision is made for an explicit version number in this syntax. If a future version is needed, it will use a different namespace. The experimental XML namespace [XML-NS] URI that MUST be used by implementations of this (dated) specification is:

```
xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
```

This namespace is also used as the prefix for algorithm identifiers used by this specification. While applications MUST support XML and XML namespaces, the use of internal entities [XML, section 4.2.1], the "`xenc`" XML namespace prefix [XML-NS, section 2] and defaulting/scoping conventions are OPTIONAL; we use these facilities to provide compact and readable examples. Additionally, the entity `&xenc;` is defined so as to provide short-hand identifiers for URIs defined in this specification. For example "`&xenc;Element`" corresponds to "http://www.w3.org/2001/04/xmlenc#Element".

This specification makes use of the XML Signature [XML-DSIG] namespace and schema definitions

```
xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
```

Los URI [URI] DEBEN cumplir con la definición de tipo [XML-Schema] y la especificación [XML-DSIG , 4.3.3.1 El atributo URI] (es decir, caracteres permitidos, escape de caracteres, soporte de esquema, etc.).`anyURI`

1.4 Agradecimientos

Se agradecen las contribuciones de los siguientes miembros del Grupo de Trabajo a esta especificación de acuerdo con las políticas de contribuyentes y la lista activa del Grupo de Trabajo .

- joseph ashwood
- Simon Blake-Wilson, Certicom
- Frank D. Cavallito, Sistemas BEA
- Eric Cohen, PricewaterhouseCoopers
- Blair Dillaway, Microsoft (Autor)
- Blake Dournaee, Seguridad RSA
- Donald Eastlake, Motorola (Editor)
- Barb Fox, Microsoft
- Christian Geuer-Pollmann, Universidad de Siegen
- Tom Gindin, IBM
- Jiandong Guo, Faos

- Phillip Hallam-Baker, Verisign
- Amir Herzberg, NewGenPay
- Merlín Hughes, Baltimore
- Federico Hirsch
- Maryann Hondo, IBM
- Takeshi Imamura, IBM (Autor)
- Mike Just, Entrust, Inc.
- Brian La Macchia, Microsoft
- Hiroshi Maruyama, IBM
- John Messing, Ley en línea
- Shivaram Mysore, Sun Microsystems
- Thane Plambeck, Verisign
- Joseph Reagle, W3C (Presidente, Editor)
- Alexei Sanin
- Jim Schaad, consultoría Soaring Hawk
- Ed Simon, XMLsec (Autor)
- Daniel Toth, Ford
- Yongge Wang, Certicom
- Steve Wiley, mi prueba

Además, agradecemos a las siguientes personas por sus comentarios durante y después de la última llamada:

- Martín Durst, W3C
- Dan Lanz, Zolera
- Susan Lesch, W3C
- David Orchard, Sistemas BEA
- Ronald Rivest, MIT

2 Descripción general y ejemplos de cifrado (no normativo)

Esta sección proporciona una descripción general y ejemplos de la sintaxis de cifrado XML. La sintaxis formal se encuentra en *Sintaxis de cifrado* (sección 3); el procesamiento específico se proporciona en *las Reglas de procesamiento* (sección 4).

Expresado en forma abreviada, el `EncryptedData`elemento tiene la siguiente estructura (donde "?" denota cero o una ocurrencia; "+" denota una o más ocurrencias; "*" denota cero o más ocurrencias; y la etiqueta de elemento vacía significa que el elemento debe ser vacío):

```
<¿Identificación de datos cifrados? ¿Tipo? ¿Tipo de Mimica? ¿Codificación?>
  <Método de cifrado/>?
  <ds:Información clave>
    <Clave cifrada>?
    <Método de acuerdo>?
    <ds:NombreClave>?
    <ds:Método de recuperación>?
    <ds:*>?
  </ds:KeyInfo>?
  <Datos cifrados>
    <ValorCifrado>?
    <¿URI de referencia de cifrado?>?
  </CipherData>
  <Propiedades de cifrado>?
</EncryptedData>
```

El `CipherData`elemento envuelve o hace referencia a los datos cifrados sin procesar. Si es envolvente, los datos cifrados sin procesar son el `CipherValue`contenido del elemento; si se hace referencia, el atributo `CipherReference`del elemento `URI`apunta a la ubicación de los datos cifrados sin procesar

2.1 Granularidad del cifrado

Consider the following fictitious payment information, which includes identification information and information appropriate to a payment method (e.g., credit card, money transfer, or electronic check):

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

This markup represents that John Smith is using his credit card with a limit of $5,000USD.

2.1.1 Encrypting an XML Element

Smith's credit card number is sensitive information! If the application wishes to keep that information confidential, it can encrypt the `CreditCard` element:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

By encrypting the entire `CreditCard` element from its start to end tags, the identity of the element itself is hidden. (An eavesdropper doesn't know whether he used a credit card or money transfer.) The `CipherData` element contains the encrypted serialization of the `CreditCard` element.

2.1.2 Encrypting XML Element Content (Elements)

As an alternative scenario, it may be useful for intermediate agents to know that John used a credit card with a particular limit, but not the card's number, issuer, and expiration date. In this case, the content (character data or children elements) of the `CreditCard` element is encrypted:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
     Type='http://www.w3.org/2001/04/xmlenc#Content'>
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

2.1.3 Encrypting XML Element Content (Character Data)

Or, consider the scenario in which all the information *except* the actual credit card number can be in the clear, including the fact that the Number element exists:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
       Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Both `CreditCard` and `Number` are in the clear, but the character data content of `Number` is encrypted.

2.1.4 Encrypting Arbitrary Data and XML Documents

If the application scenario requires all of the information to be encrypted, the whole document is encrypted as an octet sequence. This applies to arbitrary data including XML documents.

```
<?xml version='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
 MimeType='text/xml'>
  <CipherData>
```

```
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
```

2.1.5 Super-Encryption: Encrypting EncryptedData

An XML document may contain zero or more `EncryptedData` elements. `EncryptedData` cannot be the parent or child of another `EncryptedData` element. However, the actual data encrypted can be anything, including `EncryptedData` and `EncryptedKey` elements (i.e., super-encryption). During super-encryption of an `EncryptedData` or `EncryptedKey` element, one must encrypt the entire element. Encrypting only the content of these elements, or encrypting selected child elements is an invalid instance under the provided schema.
For example, consider the following:

```
<pay:PaymentInfo xmlns:pay='http://example.org/paymentv2'>
  <EncryptedData Id='ED1' xmlns='http://www.w3.org/2001/04/xmlenc#'
   Type='http://www.w3.org/2001/04/xmlenc#Element'>
    <CipherData>
      <CipherValue>originalEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

A valid super-encryption of "`//xenc:EncryptedData[@Id='ED1']`" would be:

```
<pay:PaymentInfo xmlns:pay='http://example.org/paymentv2'>
  <EncryptedData Id='ED2' xmlns='http://www.w3.org/2001/04/xmlenc#'
   Type='http://www.w3.org/2001/04/xmlenc#Element'>
    <CipherData>
      <CipherValue>newEncryptedData</CipherValue>
    </CipherData>
  </EncryptedData>
</pay:PaymentInfo>
```

where the `CipherValue` content of '`newEncryptedData`' is the base64 encoding of the encrypted octet sequence resulting from encrypting the `EncryptedData` element with `Id='ED1'`.

2.2 `EncryptedData` and `EncryptedKey` Usage

2.2.1 `EncryptedData` with Symmetric Key (`KeyName`)

```
[s1] <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
      Type='http://www.w3.org/2001/04/xmlenc#Element'/>
[s2]   <EncryptionMethod
        Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
[s3]   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[s4]     <ds:KeyName>John Smith</ds:KeyName>
[s5]   </ds:KeyInfo>
[s6]   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[s7] </EncryptedData>
```

`[s1]` The type of data encrypted may be represented as an attribute value to aid in decryption and subsequent processing. In this case, the data encrypted was an 'element'. Other alternatives include 'content' of an element, or an external octet sequence which can also be identified via the `MimeType` and `Encoding` attributes.

`[s2]` This (3DES CBC) is a symmetric key cipher.

`[s4]` The symmetric key has an associated name "John Smith".

`[s6]` `CipherData` contains a `CipherValue`, which is a base64 encoded octet sequence. Alternately, it could contain a `CipherReference`, which is a URI reference along with transforms necessary to obtain the encrypted data as an octet sequence

2.2.2 `EncryptedKey` (`ReferenceList`, `ds:RetrievalMethod`, `CarriedKeyName`)

The following `EncryptedData` structure is very similar to the one above, except this time the key is referenced using a `ds:RetrievalMethod`:

```
[t01] <EncryptedData Id='ED'
       xmlns='http://www.w3.org/2001/04/xmlenc#'>
[t02]   <EncryptionMethod
         Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc'/>
[t03]   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t04]     <ds:RetrievalMethod URI='#EK'
```

```
            Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
[t05]     <ds:KeyName>Sally Doe</ds:KeyName>
[t06]   </ds:KeyInfo>
[t07]   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
[t08] </EncryptedData>
```

`[t02]` This (AES-128-CBC) is a symmetric key cipher.

`[t04]` `ds:RetrievalMethod` is used to indicate the location of a key with type `&xenc;EncryptedKey`. The (AES) key is located at '#EK'.

`[t05]` `ds:KeyName` provides an alternative method of identifying the key needed to decrypt the `CipherData`. Either or both the `ds:KeyName` and `ds:KeyRetrievalMethod` could be used to identify the same key.

Within the same XML document, there existed an `EncryptedKey` structure that was referenced within `[t04]`:

```
[t09] <EncryptedKey Id='EK' xmlns='http://www.w3.org/2001/04/xmlenc#'>
[t10]   <EncryptionMethod
         Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
[t11]   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
[t12]     <ds:KeyName>John Smith</ds:KeyName>
[t13]   </ds:KeyInfo>
[t14]   <CipherData><CipherValue>xyzabc</CipherValue></CipherData>
[t15]   <ReferenceList>
[t16]     <DataReference URI='#ED'/>
[t17]   </ReferenceList>
[t18]   <CarriedKeyName>Sally Doe</CarriedKeyName>
[t19] </EncryptedKey>
```

`[t09]` The `EncryptedKey` element is similar to the `EncryptedData` element except that the data encrypted is always a key value.

`[t10]` The `EncryptionMethod` is the RSA public key algorithm.

`[t12]` `ds:KeyName` of "John Smith" is a property of the key necessary for decrypting (using RSA) the `CipherData`.

`[t14]` The `CipherData`'s `CipherValue` is an octet sequence that is processed (serialized, encrypted, and encoded) by a referring encrypted object's `EncryptionMethod`. (Note, an EncryptedKey's `EncryptionMethod` is the algorithm used to encrypt these octets and does not speak about what type of octets they are.)

`[t15-17]` A `ReferenceList` identifies the encrypted objects (`DataReference` and `KeyReference`) encrypted with this key. The `ReferenceList` contains a list of references to data encrypted by the symmetric key carried within this structure.

`[t18]` The `CarriedKeyName` element is used to identify the encrypted key value which may be referenced by the `KeyName` element in `ds:KeyInfo`. (Since ID attribute values must be unique to a document,`CarriedKeyName` can indicate that several `EncryptedKey` structures contain the same key value encrypted for different recipients.)

3 Encryption Syntax

This section provides a detailed description of the syntax and features for XML Encryption. Features described in this section MUST be implemented unless otherwise noted. The syntax is defined via [XML-Schema] with the following XML preamble, declaration, internal entity, and import:

```
Schema Definition:

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schema  PUBLIC "-//W3C//DTD XMLSchema 200102//EN"
 "http://www.w3.org/2001/XMLSchema.dtd"
 [
   <!ATTLIST schema
     xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
     xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'>
   <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'>
   <!ENTITY % p ''>
   <!ENTITY % s ''>
  ]>

<schema xmlns='http://www.w3.org/2001/XMLSchema' version='1.0'
        xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
        xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
        targetNamespace='http://www.w3.org/2001/04/xmlenc#'
        elementFormDefault='qualified'>

  <import namespace='http://www.w3.org/2000/09/xmldsig#'
          schemaLocation='http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd'/>
```

3.1 The `EncryptedType` Element

`EncryptedType` is the abstract type from which `EncryptedData` and `EncryptedKey` are derived. While these two latter element types are very similar with respect to their content models, a syntactical distinction is useful to processing. Implementation MUST generate laxly schema valid [XML-schema] `EncryptedData` or `EncryptedKey` as specified by the subsequent schema declarations. (Note the laxly schema valid generation means that the content permitted by `xsd:ANY` need not be valid.) Implementations SHOULD create these XML structures (`EncryptedType` elements and their descendents/content) in Normalization Form C [NFC, NFC-Corrigendum].

```
Schema Definition:

<complexType name='EncryptedType' abstract='true'>
  <sequence>
    <element name='EncryptionMethod' type='xenc:EncryptionMethodType'
             minOccurs='0'/>
    <element ref='ds:KeyInfo' minOccurs='0'/>
    <element ref='xenc:CipherData'/>
    <element ref='xenc:EncryptionProperties' minOccurs='0'/>
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
  <attribute name='Type' type='anyURI' use='optional'/>
  <attribute name='MimeType' type='string' use='optional'/>
  <attribute name='Encoding' type='anyURI' use='optional'/>
 </complexType>
```

`EncryptionMethod` is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

`ds:KeyInfo` is an optional element, defined by [XML-DSIG], that carries information about the key used to encrypt the data. Subsequent sections of this specification define new elements that may appear as children of `ds:KeyInfo`.

`CipherData` is a mandatory element that contains the `CipherValue` or `CipherReference` with the encrypted data.

`EncryptionProperties` can contain additional information concerning the generation of the `EncryptedType` (e.g., date/time stamp).

`Id` is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

`Type` is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for mandatory processing described in *Processing Rules: Decryption* (section 4.2). If the `EncryptedData` element contains data of `Type` 'element' or element 'content', and replaces that data in an XML document context, it is strongly recommended the `Type` attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

`MimeType` is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [MIME]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer `Encoding` may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the `MimeType` as 'image/png'. This attribute is purely advisory; no validation of the `MimeType` information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the `Type` attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text.

3.2 The EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

```
Schema Definition:

<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

The permitted child elements of the `EncryptionMethod` are determined by the specific value of the `Algorithm` attribute URI, and the `KeySize` child element is always permitted. For example, the RSA-OAEP algorithm (section

5.4.2) uses the ds:DigestMethod and OAEPparams elements. (We rely upon the ANY schema construct because it is not possible to specify element content based on the value of an attribute.)

La presencia de cualquier elemento secundario EncryptionMethod que no esté permitido por el algoritmo o la presencia de un KeySizeelemento secundario inconsistente con el algoritmo DEBE tratarse como un error. (Todos los URI de algoritmo especificados en este documento implican un tamaño de clave, pero esto no es cierto en general. Los algoritmos de cifrado de flujo más populares utilizan claves de tamaño variable).

3.3 El CipherDataelemento

Es CipherDataun elemento obligatorio que proporciona los datos cifrados. Debe contener la secuencia de octetos cifrados como texto codificado en base64 del CipherValueelemento o proporcionar una referencia a una ubicación externa que contenga la secuencia de octetos cifrados a través del CipherReferenceelemento.

```
Definición del esquema:

<elemento nombre=' CipherData' tipo=' xenc:CipherDataType'/>
<nombre de tipo complejo=' CipherDataType'>
   <elección>
     <elemento nombre=' CipherValue' tipo='base64Binary'/>
     <elemento ref=' xenc:CipherReference'/>
   </elección>
 </tipocomplejo>
```

3.3.1 El CipherReferenceelemento

Si CipherValueno se suministra directamente, CipherReferenceidentifica una fuente que, cuando se procesa, produce la secuencia de octetos cifrada.

El valor real se obtiene de la siguiente manera. Contiene CipherReference URIun identificador al que se le ha desreferenciado. Si el CipherReferenceelemento contiene una secuencia OPCIONAL de Transforms, los datos resultantes de desreferenciar el URI se transforman según lo especificado para producir el valor de cifrado deseado. Por ejemplo, si el valor está codificado en base64 dentro de un documento XML; las transformaciones podrían especificar una expresión XPath seguida de una decodificación base64 para extraer los octetos.

La sintaxis de URIy Transformses similar a la de [XML-DSIG]. Sin embargo, existe una diferencia entre el procesamiento de firma y cifrado. En [XML-DSIG], tanto el procesamiento de generación como el de validación comienzan con los mismos datos de origen y realizan esa transformación en el mismo orden. En el cifrado, el descifrador sólo tiene los datos cifrados y las transformaciones especificadas se enumeran para el descifrador, en el orden necesario para obtener los octetos. En consecuencia, debido a que tiene una semántica diferente, Transformsestá en el &xenc;espacio de nombres.

Por ejemplo, si el valor de cifrado relevante se captura dentro de un CipherValueelemento dentro de un documento XML diferente, CipherReferencepodría tener el siguiente aspecto:

```
<CipherReference URI="http://www.example.com/CipherValues.xml">
  <Transformaciones>
    <ds:Transformar
     Algoritmo="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <ds:XPath xmlns:rep="http://www.example.org/repositorio">
          self::text()[padre::rep:CipherValue[@Id="ejemplo1"]]
        </ds:XPath>
    </ds:Transformar>
    <ds:Algoritmo de transformación="http://www.w3.org/2000/09/xmldsig#base64"/>
  </Transforma>
</CipherReference>
```

Las implementaciones DEBEN admitir la CipherReferencefunción y la misma codificación URI, desreferenciación, esquema y códigos de respuesta HTTP que los de [XML-DSIG]. La Transformcaracterística y los algoritmos de transformación particulares son OPCIONALES.

```
Definición del esquema:

<elemento nombre=' CipherReference' tipo=' xenc:CipherReferenceType'/>
 <nombre de tipo complejo=' CipherReferenceType'>
     <secuencia>
       <elemento nombre='Transforms' tipo='xenc:TransformsType' minOccurs='0'/>
     </secuencia>
     <nombre de atributo='URI' tipo='cualquierURI' uso='requerido'/>
 </tipocomplejo>

  <nombre de tipo complejo='Tipo de transformación'>
     <secuencia>
       <elemento ref='ds:Transform' maxOccurs='ilimitado'/>
```

```
    </secuencia>
  </tipocomplejo>
```

3.4 El `EncryptedData`elemento

El `EncryptedData`elemento es el elemento central de la sintaxis. Su `CipherData`elemento secundario no solo contiene los datos cifrados, sino que también es el elemento que reemplaza al elemento cifrado o sirve como raíz del nuevo documento.

```
Definición del esquema:

<elemento nombre=' EncryptedData' tipo=' xenc:EncryptedDataType'/>
<nombre de tipo complejo=' EncryptedDataType'>
  <Contenido complejo>
   <base de extensión=' xenc:EncryptedType'>
   </extensión>
  </complexContent>
</tipocomplejo>
```

3.5 Extensiones al`ds:KeyInfo` elemento

Hay tres formas de `CipherData`proporcionar el material de claves necesario para descifrar:

1. El elemento `EncryptedData`o `EncryptedKey`especifica el material de clave asociado a través de un elemento secundario de `ds:KeyInfo`. Todos los elementos secundarios de ds: `KeyInfo`especificados en [XML-DSIG] PUEDEN usarse como calificados:
   1. La compatibilidad con `ds:KeyValue`es OPCIONAL y puede usarse para transportar claves públicas, como los valores clave Diffie-Hellman (sección 5.5.1). (Obviamente NO SE RECOMIENDA incluir la clave de descifrado de texto plano, ya sea una clave privada o secreta).
   2. Se RECOMIENDA el soporte de `ds:KeyName`para hacer referencia a un .`EncryptedKey CarriedKeyName`
   3. `ds:RetrievalMethod`Se REQUIERE soporte para el mismo documento .

   Además, proporcionamos dos elementos secundarios adicionales: las aplicaciones DEBEN ser compatibles `EncryptedKey` (sección 3.5.1) y PUEDEN ser compatibles `AgreementMethod`(sección 5.5).
2. Un elemento separado (no dentro de `ds:KeyInfo`) `EncryptedKey`puede especificar el `EncryptedData`o `EncryptedKey`al cual se aplicará su clave descifrada a través de `DataReference`o `KeyReference`(sección 3.6).
3. El material de claves lo puede determinar el destinatario según el contexto de la aplicación y, por lo tanto, no es necesario mencionarlo explícitamente en el XML transmitido.

3.5.1 El `EncryptedKey`elemento

Identificador
`Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"`

(This can be used within a `ds:RetrievalMethod` element to identify the referent's type.)

The `EncryptedKey` element is used to transport encryption keys from the originator to a known recipient(s). It may be used as a stand-alone XML document, be placed within an application document, or appear inside an `EncryptedData` element as a child of a `ds:KeyInfo` element. The key value is always encrypted to the recipient(s). When `EncryptedKey` is decrypted the resulting octets are made available to the `EncryptionMethod` algorithm without any additional processing.

```
Schema Definition:

<element name='EncryptedKey' type='xenc:EncryptedKeyType'/>
<complexType name='EncryptedKeyType'>
  <complexContent>
    <extension base='xenc:EncryptedType'>
      <sequence>
        <element ref='xenc:ReferenceList' minOccurs='0'/>
        <element name='CarriedKeyName' type='string' minOccurs='0'/>
      </sequence>
      <attribute name='Recipient' type='string' use='optional'/>
    </extension>
  </complexContent>
</complexType>
```

`ReferenceList` is an optional element containing pointers to data and keys encrypted using this key. The reference list may contain multiple references to `EncryptedKey` and `EncryptedData` elements. This is done using `KeyReference` and `DataReference` elements respectively. These are defined below.

`CarriedKeyName` is an optional element for associating a user readable name with the key value. This may then be used to reference the key using the `ds:KeyName` element within `ds:KeyInfo`. The same `CarriedKeyName` label, unlike an ID type, may occur multiple times within a single document. The value of the key is to be the same in all `EncryptedKey` elements identified with the same `CarriedKeyName` label within a single XML document. Note that because whitespace is significant in the value of the `ds:KeyName` element, whitespace is also significant in the value of the `CarriedKeyName` element.

`Recipient` is an optional attribute that contains a hint as to which recipient this encrypted key value is intended for. Its contents are application dependent.

The `Type` attribute inheritted from `EncryptedType` can be used to further specify the type of the encrypted key if the `EncryptionMethod Algorithm` does not define a unambiguous encoding/representation. (Note, all the algorithms in this specification have an unambigous representation for their associated key structures.)

3.5.2 The `ds:RetrievalMethod` Element

The `ds:RetrievalMethod` [XML-DSIG]with a `Type` of '`http://www.w3.org/2001/04/xmlenc#EncryptedKey`' provides a way to express a link to an `EncryptedKey` element containing the key needed to decrypt the `CipherData` associated with an `EncryptedData` or `EncryptedKey` element. The `ds:RetrievalMethod` with this type is always a child of the `ds:KeyInfo` element and may appear multiple times. If there is more than one instance of a `ds:RetrievalMethod` in a `ds:KeyInfo` of this type, then the `EncryptedKey` objects referred to must contain the same key value, possibly encrypted in different ways or for different recipients.

```
Schema Definition:

<!--
    <attribute name='Type' type='anyURI' use='optional'
    fixed='http://www.w3.org/2001/04/xmlenc#EncryptedKey' />
-->
```

3.6 The `ReferenceList` Element

`ReferenceList` is an element that contains pointers from a key value of an `EncryptedKey` to items encrypted by that key value (`EncryptedData` or `EncryptedKey` elements).

```
Schema Definition:

<element name='ReferenceList'>
  <complexType>
    <choice minOccurs='1' maxOccurs='unbounded'>
      <element name='DataReference' type='xenc:ReferenceType'/>
      <element name='KeyReference' type='xenc:ReferenceType'/>
    </choice>
  </complexType>
</element>

<complexType name='ReferenceType'>
  <sequence>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='URI' type='anyURI' use='required'/>
</complexType>
```

`DataReference` elements are used to refer to `EncryptedData` elements that were encrypted using the key defined in the enclosing `EncryptedKey` element. Multiple `DataReference` elements can occur if multiple `EncryptedData` elements exist that are encrypted by the same key.

`KeyReference` elements are used to refer to `EncryptedKey` elements that were encrypted using the key defined in the enclosing `EncryptedKey` element. Multiple `KeyReference` elements can occur if multiple `EncryptedKey` elements exist that are encrypted by the same key.

For both types of references one may optionally specify child elements to aid the recipient in retrieving the `EncryptedKey` and/or `EncryptedData` elements. These could include information such as XPath transforms, decompression transforms, or information on how to retrieve the elements from a document storage facility. For example:

```
<ReferenceList>
  <DataReference URI="#invoice34">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            self::xenc:EncryptedData[@Id="example1"]
        </ds:XPath>
```

```
      </ds:Transform>
    </ds:Transforms>
  </DataReference>
</ReferenceList>
```

3.7 The `EncryptionProperties` Element

Identifier
    `Type="http://www.w3.org/2001/04/xmlenc#EncryptionProperties"`

    (This can be used within a `ds:Reference` element to identify the referent's type.)

Additional information items concerning the generation of the `EncryptedData` or `EncryptedKey` can be placed in an `EncryptionProperty` element (e.g., date/time stamp or the serial number of cryptographic hardware used during encryption). The `Target` attribute identifies the `EncryptedType` structure being described. `anyAttribute` permits the inclusion of attributes from the XML namespace to be included (i.e., `xml:space`, `xml:lang`, and `xml:base`).

```
Schema Definition:

<element name='EncryptionProperties' type='xenc:EncryptionPropertiesType'/>
<complexType name='EncryptionPropertiesType'>
  <sequence>
    <element ref='xenc:EncryptionProperty' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
</complexType>

  <element name='EncryptionProperty' type='xenc:EncryptionPropertyType'/>
  <complexType name='EncryptionPropertyType' mixed='true'>
    <choice maxOccurs='unbounded'>
      <any namespace='##other' processContents='lax'/>
    </choice>
    <attribute name='Target' type='anyURI' use='optional'/>
    <attribute name='Id' type='ID' use='optional'/>
    <anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
  </complexType>
```

4 Processing Rules

This section describes the operations to be performed as part of encryption and decryption processing by implementations of this specification. The conformance requirements are specified over the following roles:

Application
    The application which makes request of an XML Encryption implementation via the provision of data and parameters necessary for its processing.
Encryptor
    An XML Encryption implementation with the role of encrypting data.
Decryptor
    An XML Encryption implementation with the role of decrypting data.

4.1 Encryption

For each data item to be encrypted as an `EncryptedData` or `EncryptedKey` (elements derived from `EncryptedType`), the **encryptor** must:

1. Select the algorithm (and parameters) to be used in encrypting this data.
2. Obtain and (optionally) represent the key.
    1. If the key is to be identified (via naming, URI, or included in a child element), construct the `ds:KeyInfo` as approriate (e.g., `ds:KeyName`, `ds:KeyValue`, `ds:RetrievalMethod`, etc.)
    2. If the key itself is to be encrypted, construct an `EncryptedKey` element by recursively applying this encryption process. The result may then be a child of `ds:KeyInfo`, or it may exist elsewhere and may be identified in the preceding step.
3. Encrypt the data
    1. If the data is an 'element' [XML, section 3] or element 'content' [XML, section 3.1], obtain the octets by serializing the data in UTF-8 as specified in [XML]. (The application MUST provide XML data in [NFC].) Serialization MAY be done by the **encryptor**. If the **encryptor** does not serialize, then the **application** MUST perform the serialization.
    2. If the data is of any other type that is not already octets, the **application** MUST serialize it as octets.
    3. Encrypt the octets using the algorithm and key from steps 1 and 2.
    4. Unless the **decryptor** will implicitly know the type of the encrypted data, the **encryptor** SHOULD provide the type for representation.

The definition of this type as bound to an identifier specifies how to obtain and interpret the plaintext octets after decryption. For example, the idenifier could indicate that the data is an instance of another application (e.g., some XML compression application) that must be further processed. Or, if the data is a simple octet sequence it MAY be described with the `MimeType` and `Encoding` attributes. For example, the data might be an XML document (`MimeType="text/xml"`), sequence of characters (`MimeType="text/plain"`), or binary image data (`MimeType="image/png"`).

4. Build the `EncryptedType` (`EncryptedData` or `EncryptedKey`) structure:

   An `EncryptedType` structure represents all of the information previously discussed including the type of the encrypted data, encryption algorithm, parameters, key, type of the encrypted data, etc.

   1. If the encrypted octet sequence obtained in step 3 is to be stored in the `CipherData` element within the `EncryptedType`, then the encrypted octet sequence is base64 encoded and inserted as the content of a `CipherValue` element.
   2. If the encrypted octet sequence is to be stored externally to the `EncryptedType` structure, then store or return the encrypted octet sequence, and represent the URI and transforms (if any) required for the decryptor to retrieve the encrypted octet sequence within a `CipherReference` element.
5. Process EncryptedData
   1. If the `Type` of the encrypted data is '[element]' or element '[content]', then the **encryptor** MUST be able to return the `EncryptedData` element to the **application**. The **application** MAY use this as the top-level element in a new XML document or insert it into another XML document, which may require a re-encoding.

      The **encryptor** SHOULD be able to replace the unencrypted 'element' or 'content' with the EncryptedData element. When an **application** requires an XML element or content to be replaced, it supplies the XML document context in addition to identifying the element or content to be replaced. The **encryptor** removes the identified element or content and inserts the `EncryptedData` element in its place.

      (Note: If the `Type` is "content" the document resulting from decryption will not be well-formed if (a) the original plaintext was not well-formed (e.g., PCDATA by itself is not well-formed) and (b) the `EncryptedData` element was previously the root element of the document)

   2. If the `Type` of the encrypted data *is not* '[element]' or element '[content]', then the **encryptor** MUST always return the `EncryptedData` element to the **application**. The **application** MAY use this as the top-level element in a new XML document or insert it into another XML document, which may require a re-encoding.

4.2 Decryption

For each `EncryptedType` derived element, (i.e., `EncryptedData` or `EncryptedKey`), to be decrypted, the **decryptor** must:

1. Process the element to determine the algorithm, parameters and `ds:KeyInfo` element to be used. If some information is omitted, the **application** MUST supply it.
2. Locate the data encryption key according to the `ds:KeyInfo` element, which may contain one or more children elements. These children have no implied processing order. If the data encryption key is encrypted, locate the corresponding key to decrypt it. (This may be a recursive step as the key-encryption key may itself be encrypted.) Or, one might retrieve the data encryption key from a local store using the provided attributes or implicit binding.
3. Decrypt the data contained in the `CipherData` element.
   1. If a `CipherValue` child element is present, then the associated text value is retrieved and base64 decoded so as to obtain the encrypted octet sequence.
   2. If a `CipherReference` child element is present, the URI and transforms (if any) are used to retrieve the encrypted octet sequence.
   3. The encrypted octet sequence is decrypted using the algorithm/parameters and key value already determined from steps 1 and 2.
4. Process decrypted data of `Type` '[element]' or element '[content]'.
   1. The cleartext octet sequence obtained in step 3 is interpreted as UTF-8 encoded character data.
   2. The **decryptor** MUST be able to return the value of `Type` and the UTF-8 encoded XML character data. The **decryptor** is NOT REQUIRED to perform validation on the serialized XML.
   3. The **decryptor** SHOULD support the ability to replace the `EncryptedData` element with the decrypted '[element]' or element '[content]' represented by the UTF-8 encoded characters. The **decryptor** is NOT REQUIRED to perform validation on the result of this replacement operation.

      The application supplies the XML document context and identifies the `EncryptedData` element being replaced. If the document into which the replacement is occurring is not UTF-8, the **decryptor** MUST transcode the UTF-8 encoded characters into the target encoding.

5. Process decrypted data if `Type` is unspecified or is *not* 'element' or element 'content'.
    1. The cleartext octet sequence obtained in **Step 3** MUST be returned to the **application** for further processing along with the `Type`, `MimeType`, and `Encoding` attribute values when specified. `MimeType` and `Encoding` are advisory. The `Type` value is normative as it may contain information necessary for the processing or interpration of the data by the application.
    2. Note, this step includes processing data decrypted from an `EncryptedKey`. The cleartext octet sequence represents a key value and is used by the application in decrypting other `EncryptedType` element(s).

4.3 XML Encryption

Encryption and decryption operations are transforms on octets. The **application** is responsible for the marshalling XML such that it can be serialized into an octet sequence, encrypted, decrypted, and be of use to the recipient.

For example, if the application wishes to canonicalize its data or encode/compress the data in an XML packaging format, the application needs to marshal the XML accordingly and identify the resulting type via the `EncryptedData` `Type` attribute. The likelihood of successful decryption and subsequent processing will be dependent on the recipient's support for the given type. Also, if the data is intended to be processed both before encryption and after decryption (e.g., XML Signature [XML-DSIG] validation or an XSLT transform) the encrypting application must be careful to preserve information necessary for that process's success.

For interoperability purposes, the following types MUST be implemented such that an implementation will be able to take as input and yield as output data matching the production rules 39 and 43 from [XML]:

element 'http://www.w3.org/2001/04/xmlenc#Element'
"[39] element ::= `EmptyElemTag` | STag content ETag"
content 'http://www.w3.org/2001/04/xmlenc#Content'
"[43] content ::= `CharData`? ((element | Reference | CDSect | PI | Comment) `CharData`?)*"

The following sections contain specifications for decrypting, replacing, and serializing XML content (i.e., `Type` 'element' or element 'content') using the [XPath] data model. These sections are non-normative and OPTIONAL to implementors of this specification, but they may be normatively referenced by and MANDATORY to other specifications that require a consistent processing for applications, such as [XML-DSIG-Decrypt].

4.3.1 A Decrypt Implementation (Non-normative)

Where *P* is the context in which the serialized XML should be parsed (a document node or element node) and *O* is the octet sequence representing UTF-8 encoded characters resulting from step 4.3 in the *Decryption Processing* (section 4.2). *Y* is node-set representing the decrypted content obtained by the following steps:

1. Let *C* be the **parsing context** of a child of *P*, which consists of the following items:
    - Prefix and namespace name of each namespace that is in scope for *P*.
    - Name and value of each general entity that is effective for the XML document causing *P*.
2. Wrap the decrypted octet stream *O* in the context *C* as specified in *Text Wrapping*.
3. Parse the wrapped octet stream as described in *The Reference Processing Model* (section 4.3.3.2) of [XML-Signature], resulting in a node-set.
4. *Y* is the node-set obtained by removing the root node, the wrapping element node, and its associated set of attribute and namespace nodes from the node-set obtained in Step 3.

4.3.2 A Decrypt and Replace Implementation (Non-normative)

Where *X* is the [XPath] node set corresponding to an XML document and e is an `EncryptedData` element node in *X*.

1. *Z* is an [XPath] node-set that identical to X except where the element node e is an `EncryptedData` element type. In which case:
    1. Decrypt e in the context of its parent node as specified in the *Decryption Implementation* (section 4.3.1) yielding *Y*, an [XPath] node set.
    2. Include *Y* in place of *e* and its descendants in *X*. Since [XPath] does not define methods of replacing node-sets from different documents, the result MUST be equivalent to replacing e with the octet stream resulting from its decryption in the serialized form of *X* and reparsing the document. However, the actual method of performing this operation is left to the implementor.

4.3.3 Serializing XML (Non-normative)

Default Namespace Considerations

In *Encrypting XML* (section 4.1, step 3.1), when serializing an XML fragment special care SHOULD be taken with respect to default namespaces. If the data will be subsequently decrypted in the context of a parent XML document then serialization can produce elements in the wrong namespace. Consider the following fragment of XML:

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted xmlns="" />
</Document>
```

Serialization of the element `ToBeEncrypted` fragment via [XML-C14N] would result in the characters "`<ToBeEncrypted></ToBeEncrypted>`" as an octet stream. The resulting encrypted document would be:

```
<Document xmlns="http://example.org/">
  <EncryptedData xmlns="...">
    <!-- Containing the encrypted
         "<ToBeEncrypted></ToBeEncrypted>" -->
  </EncryptedData>
</Document>
```

Decrypting and replacing the `EncryptedData` within this document would produce the following incorrect result:

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted/>
</Document>
```

This problem arises because most XML serializations assume that the serialized data will be parsed directly in a context where there is no default namespace declaration. Consequently, they do not redundantly declare the empty default namespace with an `xmlns=""`. If, however, the serialized data is parsed in a context where a default namespace declaration is in scope (e.g., the parsing context of a *A Decrypt Implementation* (section 4.3.1)), then it may affect the interpretation of the serialized data.

To solve this problem, a canonicalization algorithm MAY be augmented as follows for use as an XML encryption serializer:

- A default namespace declaration with an empty value (i.e., `xmlns=""`) SHOULD be emitted where it would normally be suppressed by the canonicalization algorithm.

While the result may not be in proper canonical form, this is harmless as the resulting octet stream will not be used directly in a [XML-Signature] signature value computation. Returning to the preceding example with our new augmentation, the `ToBeEncrypted` element would be serialized as follows:

```
<ToBeEncrypted xmlns=""></ToBeEncrypted>
```

When processed in the context of the parent document, this serialized fragment will be parsed and interpreted correctly.

This augmentation can be retroactively applied to an existing canonicalization implementation by canonicalizing each apex node and its descendants from the node set, inserting `xmlns=""` at the appropriate points, and concatenating the resulting octet streams.

XML Attribute Considerations

Similar attention between the relationship of a fragment and the context into which it is being inserted should be given to the `xml:base`, `xml:lang`, and `xml:space` attributes as mentioned in the *Security Considerations* of [XML-exc-C14N]. For example, if the element:

```
<Bongo href="example.xml"/>
```

is taken from a context and serialized with no `xml:base` [XML-Base] attribute and parsed in the context of the element:

```
<Baz xml:base="http://example.org/"/>
```

the result will be:

```
<Baz xml:base="http://example.org/"><Bongo href="example.xml"/></Baz>
```

`Bongo`'s `href` is subsequently interpreted as "`http://example.org/example.xml`". If this is not the correct URI, `Bongo` should have been serialized with its own `xml:base` attribute.

Desafortunadamente, la recomendación de que se emita un valor vacío para separar el espacio de nombres predeterminado del fragmento del contexto en el que se inserta no se puede realizar para los atributos `xml:base`y `xml:space`. (*El error 41* de la errata de especificación XML 1.0 de segunda edición aclara que un valor de cadena vacío del atributo `xml:lang` *se* considera como si "no hubiera información de idioma disponible, como si `xml:lang`no se hubiera especificado".) La interpretación de un valor vacío para los atributos `xml:base`o `xml:space`no están definidos o mantienen el valor contextual. En consecuencia, las aplicaciones DEBEN garantizar (1) que los fragmentos que se van a cifrar no dependan de atributos XML, o (2) si son dependientes y se pretende que el documento resultante sea válido [XML], la definición del fragmento *permite* la presencia del atributos y que los atributos tengan valores no vacíos.

4.3.4 Ajuste de texto (no normativo)

Esta sección especifica el proceso para ajustar texto en un contexto de análisis determinado. El proceso se basa en la propuesta de Richard Tobin [Tobin] para construir el conjunto de información [XML-Infoset] de una entidad externa.

El proceso consta de los siguientes pasos:

1. Si el contexto de análisis contiene entidades generales, emita una declaración de tipo de documento que proporcione declaraciones de entidades.
2. Emita una `dummy`etiqueta de inicio de elemento con atributos de declaración de espacio de nombres que declaren todos los espacios de nombres en el contexto de análisis.
3. Emitir el texto.
4. Emitir una `dummy`etiqueta final de elemento.

En los pasos anteriores, la declaración del tipo de documento y `dummy` las etiquetas de elementos DEBEN codificarse en UTF-8.

Considere el siguiente documento que contiene un `EncryptedData` elemento:

```
<!DOCTYPE Documento [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<Documento xmlns="http://ejemplo.org/">
  <foo:Cuerpo xmlns:foo="http://example.org/foo">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Escriba=" http://www.w3.org/2001/04/xmlenc#Element ">
      ...
    </EncryptedData>
  </foo:Cuerpo>
</Documento>
```

Si el `EncryptedData`elemento se alimenta y se descifra en el texto " `<One><foo:Two/></One>`", entonces el formato ajustado es el siguiente:

```
<!DOCTYPE ficticio [
  <!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<dummy xmlns="http://ejemplo.org/"
       xmlns:foo="http://example.org/foo"><Uno><foo:Two/></Uno></dummy>
```

5. Algoritmos

Esta sección analiza los algoritmos utilizados con la especificación de cifrado XML. Las entradas contienen el identificador que se utilizará como valor del `Algorithm`atributo del `EncryptionMethod`elemento u otro elemento que represente el rol del algoritmo, una referencia a la especificación formal, definiciones para la representación de claves y los resultados de las operaciones criptográficas cuando corresponda, y información general. comentarios de aplicabilidad.

5.1 Identificadores de algoritmos y requisitos de implementación

Todos los algoritmos enumerados a continuación tienen parámetros implícitos según su función. Por ejemplo, los datos que se van a cifrar o descifrar, el material de claves y la dirección de operación (cifrado o descifrado) de los algoritmos de cifrado. Cualquier parámetro adicional explícito de un algoritmo aparece como elementos de contenido dentro del elemento. Dichos elementos secundarios de parámetros tienen nombres de elementos descriptivos, que frecuentemente son específicos del algoritmo, y DEBEN estar en el mismo espacio de nombres que esta especificación de cifrado XML, la especificación de firma XML o en un espacio de nombres específico del algoritmo. Un ejemplo de un parámetro tan explícito podría ser un nonce (cantidad única) proporcionado a un algoritmo de acuerdo clave.

Esta especificación define un conjunto de algoritmos, sus URI y requisitos de implementación. Los niveles de requisitos especificados, como "REQUERIDO" u "OPCIONAL", se refieren a la implementación, no al uso. Además, el mecanismo es extensible y se pueden utilizar algoritmos alternativos.

Tabla de algoritmos

La siguiente tabla enumera las categorías de algoritmos. Dentro de cada categoría, se proporciona un nombre breve, el nivel de requisito de implementación y un URI de identificación para cada algoritmo.

Cifrado de bloques

1. TRIPLEDES REQUERIDOS
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc
2. AES-128 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes128-cbc
3. AES-256 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes256-cbc
4. OPCIONAL AES-192
   http://www.w3.org/2001/04/xmlenc#aes192-cbc

Cifrado de flujo

1. none
   A continuación se proporcionan sintaxis y recomendaciones para admitir algoritmos especificados por el usuario.

Transporte clave

1. REQUERIDO RSA-v1.5
   http://www.w3.org/2001/04/xmlenc#rsa-1_5
2. REQUERIDO RSA-OAEP
   http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p

Acuerdo clave

1. OPCIONAL Diffie-Hellman
   http://www.w3.org/2001/04/xmlenc#dh

Envoltura de clave simétrica

1. TRIPLEDES REQUERIDOS KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-tripledes
2. REQUERIDO AES-128 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes128
3. REQUERIDO AES-256 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes256
4. OPCIONAL AES-192 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes192

Resumen del mensaje

1. SHA1 REQUERIDO
   http://www.w3.org/2000/09/xmldsig#sha1
2. RECOMENDADO SHA256
   http://www.w3.org/2001/04/xmlenc#sha256
3. SHA512 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#sha512
4. RIPEMD-160 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#ripemd160

Autenticación de mensajes

1. Firma digital XML RECOMENDADA
   http://www.w3.org/2000/09/xmldsig#

Canonicalización

1. XML canónico OPCIONAL (omite comentarios)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315

2. XML canónico OPCIONAL con comentarios
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
3. OPCIONAL Canonicalización XML exclusiva (omite comentarios)
   http://www.w3.org/2001/10/xml-exc-c14n#
4. OPCIONAL Canonicalización XML exclusiva con comentarios
   http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Codificación

1. REQUERIDO base64
   http://www.w3.org/2000/09/xmldsig#base64

5.2 Algoritmos de cifrado de bloques

Los algoritmos de cifrado de bloques están diseñados para cifrar y descifrar datos en bloques de varios octetos de tamaño fijo. Sus identificadores aparecen como el valor de los `Algorithm`atributos de `EncryptionMethod`los elementos hijos de `EncryptedData`.

Los algoritmos de cifrado de bloques toman, como argumentos implícitos, los datos que se van a cifrar o descifrar, el material de clave y su dirección de operación. Para todos estos algoritmos especificados a continuación, se requiere un vector de inicialización (IV) codificado con el texto cifrado. Para los algoritmos de cifrado de bloques especificados por el usuario, el IV, si lo hubiera, podría especificarse junto con los datos cifrados, como un elemento de contenido del algoritmo o en cualquier otro lugar.

El IV está codificado con y antes del texto cifrado para los algoritmos siguientes para facilitar la disponibilidad del código de descifrado y enfatizar su asociación con el texto cifrado. Las buenas prácticas criptográficas requieren que se utilice un IV diferente para cada cifrado.

Relleno

Dado que los datos que se cifran son un número arbitrario de octetos, es posible que no sean un múltiplo del tamaño del bloque. Esto se resuelve rellenando el texto sin formato hasta el tamaño del bloque antes del cifrado y deshaciendo el relleno después del descifrado. El algoritmo de relleno consiste en calcular el número de octetos más pequeño distinto de cero, por ejemplo `N`, que debe añadirse al texto sin formato para que sea un múltiplo del tamaño del bloque. Supondremos que el tamaño del bloque es `B` de octetos, por lo que `N`está en el rango de 1 a `B`. Rellene añadiendo al texto sin formato un sufijo de `N-1`bytes de relleno arbitrarios y un byte final cuyo valor sea `N`. Al descifrar, simplemente tome el último byte y, después de verificarlo, elimine esa cantidad de bytes del final del texto cifrado descifrado.

Por ejemplo, supongamos un tamaño de bloque de 8 bytes y un texto sin formato de `0x616263`. El texto sin formato acolchado estaría entonces `0x616263????????05`donde "??" Los bytes pueden tener cualquier valor. De manera similar, el texto sin formato de `0x2122232425262728`se rellenaría con `0x2122232425262728??????????????08`.

5.2.1 Triple DES

Identificador:
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc (REQUERIDO)

ANSI X9.52 [TRIPLEDES] especifica tres operaciones FIPS 46-3 [DES] secuenciales. El cifrado XML TRIPLEDES consta de un cifrado DES, un descifrado DES y un cifrado DES utilizado en el modo Cipher Block Chaining (CBC) con 192 bits de clave y un vector de inicialización (IV) de 64 bits. De los bits clave, los primeros 64 bits se utilizan en la primera operación DES, los segundos 64 bits en la operación DES intermedia y los terceros 64 bits en la última operación DES.

Nota: Cada uno de estos 64 bits de clave contiene 56 bits efectivos y 8 bits de paridad. Por tanto, sólo hay 168 bits operativos de los 192 que se transportan para una clave TRIPLEDES. (Dependiendo del criterio utilizado para el análisis, se puede pensar que la fuerza efectiva de la clave es de 112 bits (debido a los ataques intermedios) o incluso menos).

El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado TRIPLEDES es el siguiente:

```
<Método de cifrado
 Algoritmo="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

5.2.2 AES

Identificador:

http://www.w3.org/2001/04/xmlenc#aes128-cbc (REQUERIDO)
http://www.w3.org/2001/04/xmlenc#aes192-cbc (OPCIONAL)
http://www.w3.org/2001/04/xmlenc#aes256-cbc (REQUERIDO)

[AES] se utiliza en el modo Cipher Block Chaining (CBC) con un vector de inicialización (IV) de 128 bits. El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado AES es el siguiente:

```
<Método de cifrado
 Algoritmo="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
```

5.3 Algoritmos de cifrado de flujo

Los algoritmos de cifrado de flujo simple generan, en función de la clave, un flujo de bytes a los que se aplica XOR con los bytes de datos de texto sin formato para producir el texto cifrado en el cifrado y con los bytes de texto cifrado para producir texto sin formato al descifrar. Normalmente se utilizan para el cifrado de datos y se especifican por el valor del `Algorithm`atributo del `EncryptionMethod`hijo de un `EncryptedData` elemento.

NOTA: Es fundamental que cada clave de cifrado de flujo simple (o clave y vector de inicialización (IV) si también se usa un IV) se use solo una vez. Si alguna vez se usa la misma clave (o clave y IV) en dos mensajes, al aplicar XOR en los dos textos cifrados, puede obtener el XOR de los dos textos sin formato. Esto suele ser muy comprometedor.

En este documento no se especifican algoritmos de cifrado de flujo específicos, pero esta sección se incluye para proporcionar pautas generales.

Stream algorithms typically use the optional `KeySize` explicit parameter. In cases where the key size is not apparent from the algorithm URI or key source, as in the use of key agreement methods, this parameter sets the key size. If the size of the key to be used is apparent and disagrees with the `KeySize` parameter, an error MUST be returned. Implementation of any stream algorithms is optional. The schema for the KeySize parameter is as follows:

```
Schema Definition:

<simpleType name='KeySizeType'>
  <restriction base="integer"/>
</simpleType>
```

5.4 Key Transport

Key Transport algorithms are public key encryption algorithms especially specified for encrypting and decrypting keys. Their identifiers appear as `Algorithm` attributes to `EncryptionMethod` elements that are children of `EncryptedKey`. `EncryptedKey` is in turn the child of a `ds:KeyInfo` element. The type of key being transported, that is to say the algorithm in which it is planned to use the transported key, is given by the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` parent of this `ds:KeyInfo` element.

(Key Transport algorithms may optionally be used to encrypt data in which case they appear directly as the `Algorithm` attribute of an `EncryptionMethod` child of an `EncryptedData` element. Because they use public key algorithms directly, Key Transport algorithms are not efficient for the transport of any amounts of data significantly larger than symmetric keys.)

The RSA v1.5 Key Transport algorithm given below are those used in conjunction with TRIPLEDES and the Cryptographic Message Syntax (CMS) of S/MIME [CMS-Algorithms]. The RSA v2 Key Transport algorithm given below is that used in conjunction with AES and CMS [AES-WRAP].

5.4.1 RSA Version 1.5

Identifier:
http://www.w3.org/2001/04/xmlenc#rsa-1_5 (REQUIRED)

The RSAES-PKCS1-v1_5 algorithm, specified in RFC 2437 [PKCS1], takes no explicit parameters. An example of an RSA Version 1.5 `EncryptionMethod` element is:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

The `CipherValue` for such an encrypted key is the base64 [MIME] encoding of the octet string computed as per RFC 2437 [PKCS1, section 7.2.1: Encryption operation]. As specified in the EME-PKCS1-v1_5 function RFC 2437 [PKCS1, section 9.1.2.1], the value input to the key transport function is as follows:

```
CRYPT ( PAD ( KEY ))
```

where the padding is of the following special form:

```
02 | PS* | 00 | key
```

where "|" is concatenation, "02" and "00" are fixed octets of the corresponding hexadecimal value, PS is a string of strong pseudo-random octets [RANDOM] at least eight octets long, containing no zero octets, and long enough that the value of the quantity being CRYPTed is one octet shorter than the RSA modulus, and "key" is the key being transported. The key is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Support of this key transport algorithm for transporting 192 bit keys is MANDATORY to implement. Support of this algorithm for transporting other keys is OPTIONAL. RSA-OAEP is RECOMMENDED for the transport of AES keys.

The resulting base64 [MIME] string is the value of the child text node of the `CipherData` element, e.g.

```
<CipherData> IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4
   t0/gyTE96639In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw=
</CipherData>
```

5.4.2 RSA-OAEP

Identifier:
http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (REQUIRED)

The RSAES-OAEP-ENCRYPT algorithm, as specified in RFC 2437 [PKCS1], takes three parameters. The two user specified parameters are a MANDATORY message digest function and an OPTIONAL encoding octet string `OAEPparams`. The message digest function is indicated by the `Algorithm` attribute of a child `ds:DigestMethod` element and the mask generation function, the third parameter, is always MGF1 with SHA1 (mgf1SHA1Identifier). Both the message digest and mask generation functions are used in the EME-OAEP-ENCODE operation as part of RSAES-OAEP-ENCRYPT. The encoding octet string is the base64 decoding of the content of an optional `OAEPparams` child element . If no `OAEPparams` child is provided, a null string is used.

```
Schema Definition:
     <!-- use these element types as children of EncryptionMethod
          when used with RSA-OAEP -->
     <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
     <element ref='ds:DigestMethod' minOccurs='0'/>
```

An example of an RSA-OAEP element is:

```
<EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
  <OAEPparams> 9lWu3Q== </OAEPparams>
  <ds:DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<EncryptionMethod>
```

La `CipherValue`clave cifrada RSA-OAEP es la codificación base64 [MIME] de la cadena de octetos calculada según RFC 2437 [PKCS1 , sección 7.1.1: Operación de cifrado]. Como se describe en la función EME-OAEP-ENCODE RFC 2437 [PKCS1 , sección 9.1.1.1], el valor ingresado a la función de transporte de claves se calcula usando la función de resumen de mensajes y la cadena especificada en los elementos `DigestMethod`y `OAEPparams` y usando la función de generación de máscaras MGF1. (con SHA1) especificado en RFC 2437. La longitud de salida deseada para EME-OAEP-ENCODE es un byte más corta que el módulo RSA.

El tamaño de la clave transportada es de 192 bits para TRIPLEDES y de 128, 192 o 256 bits para AES. Las implementaciones DEBEN implementar RSA-OAEP para el transporte de claves de 128 y 256 bits. PUEDEN implementar RSA-OAEP para el transporte de otras claves.

5.5 Acuerdo clave

Un algoritmo de acuerdo de clave proporciona la derivación de una clave secreta compartida basada en un secreto compartido calculado a partir de ciertos tipos de claves públicas compatibles tanto del remitente como del destinatario. La información del creador para determinar el secreto se indica mediante un `OriginatorKeyInfo`parámetro opcional secundario de un `AgreementMethod`elemento, mientras que el asociado con el destinatario se indica mediante un parámetro opcional `RecipientKeyInfo`. Una clave compartida se deriva de este secreto compartido mediante un método determinado por el algoritmo de Acuerdo de Clave.

Nota: XML Encryption no proporciona un protocolo de negociación de acuerdos de claves en línea. `AgreementMethod`El creador puede utilizar el elemento para identificar las claves y el procedimiento computacional

que se utilizaron para obtener una clave de cifrado compartida. El método utilizado para obtener o seleccionar las claves o algoritmo utilizado para el cálculo del acuerdo está fuera del alcance de esta especificación.

El `AgreementMethod`elemento aparece como contenido de a `ds:KeyInfo`ya que, al igual que otros `ds:KeyInfo`elementos secundarios, produce una clave. Éste, `ds:KeyInfo`a su vez, es hijo de un elemento `EncryptedData`o `EncryptedKey`. El `Algorithm`atributo y `KeySize`el elemento secundario del `EncryptionMethod`elemento bajo este `EncryptedData` o `EncryptedKey`elemento son parámetros implícitos para el cálculo del acuerdo clave. En los casos en que este `EncryptionMethod` algoritmo URI sea insuficiente para determinar la longitud de la clave, `KeySize`DEBE haberse incluido. Además, el remitente puede colocar un `KA-Nonce`elemento debajo `AgreementMethod`para garantizar que se genere material de claves diferente incluso para acuerdos repetidos que utilicen las mismas claves públicas de remitente y destinatario. Por ejemplo:

```
<Datos cifrados>
   <Algoritmo de método de cifrado="Ejemplo:Bloque/Alg"
     <Tamaño de clave>80</Tamaño de clave>
   </Método de cifrado>
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <Algoritmo Método de Acuerdo="ejemplo:Acuerdo/Algoritmo">
       <KA-Nonce>Zm9v</KA-Nonce>
       <ds:Método de resumen
       Algoritmo="http://www.w3.org/2001/04/xmlenc#sha1"/>
      <Información de clave del originador>
         <ds:ValorClave>....</ds:ValorClave>
       </OriginatorKeyInfo>
       <Información de clave del destinatario>
         <ds:ValorClave>....</ds:ValorClave>
       </RecipientKeyInfo>
     </Método de acuerdo>
   </ds:Información clave>
   <CipherData>...</CipherData>
</EncryptedData>
```

Si la clave acordada se utiliza para envolver una clave, en lugar de datos como se indica arriba, `AgreementMethod`aparecerá dentro de `ds:KeyInfo`un `EncryptedKey`elemento.

El esquema `AgreementMethod`es el siguiente:

```
  Definición del esquema:

  <elemento nombre="AgreementMethod" type="xenc:AgreementMethodType"/>
  <complexType nombre="AgreementMethodType" mixto="verdadero">
    <secuencia>
      <elemento nombre="KA-Nonce" minOccurs="0" tipo="base64Binary"/>
      <!-- <elemento ref="ds:DigestMethod" minOccurs="0"/> -->
      <cualquier espacio de nombre="##other" minOccurs="0" maxOccurs="ilimitado"/>
      <elemento nombre="OriginatorKeyInfo" minOccurs="0"
               tipo="ds:KeyInfoType"/>
      <elemento nombre="RecipientKeyInfo" minOccurs="0"
               tipo="ds:KeyInfoType"/>
    </secuencia>
    <atributo nombre="Algoritmo" tipo="cualquierURI" uso="requerido"/>
  </tipocomplejo>
```

5.5.1 Valores clave de Diffie-Hellman

Identificador:

http://www.w3.org/2001/04/xmlenc#DHKeyValue (OPCIONAL)

Las claves Diffie-Hellman pueden aparecer directamente dentro de `KeyValue` los elementos u obtenerse mediante `ds:RetrievalMethod`recuperaciones, además de aparecer en certificados y similares. El identificador anterior se puede utilizar como el valor del `Type`atributo de `Reference`o `ds:RetrievalMethod`elementos.

Como se especifica en [ESDH], una clave pública DH consta de hasta seis cantidades, dos números primos grandes p y q, un "generador" g, la clave pública y los parámetros de validación "seed" y "pgenCounter". Estos se relacionan de la siguiente manera: La clave pública = (g**x mod p) donde x es la clave privada correspondiente; p = j*q + 1 donde j >= 2. "seed" y "pgenCounter" son opcionales y se pueden utilizar para determinar si la clave Diffie-Hellman se ha generado de conformidad con el algoritmo especificado en [ESDH] . Debido a que los números primos y el generador se pueden compartir de forma segura entre muchas claves DH, es posible que se conozcan desde el entorno de la aplicación y son opcionales. El esquema para a `DHKeyValue`es el siguiente:

```
   Schema:

  <element name="DHKeyValue" type="xenc:DHKeyValueType"/>
  <complexType name="DHKeyValueType">
     <sequence>
```

```
        <sequence minOccurs="0">
           <element name="P" type="ds:CryptoBinary"/>
           <element name="Q" type="ds:CryptoBinary"/>
           <element name="Generator"type="ds:CryptoBinary"/>
        </sequence>
        <element name="Public" type="ds:CryptoBinary"/>
        <sequence minOccurs="0">
           <element name="seed" type="ds:CryptoBinary"/>
           <element name="pgenCounter" type="ds:CryptoBinary"/>
        </sequence>
     </sequence>
  </complexType>
```

5.5.2 Acuerdo clave Diffie-Hellman

Identificador:

http://www.w3.org/2001/04/xmlenc#dh (OPCIONAL)

The Diffie-Hellman (DH) key agreement protocol [ESDH] involves the derivation of shared secret information based on compatible DH keys from the sender and recipient. Two DH public keys are compatible if they have the same prime and generator. If, for the second one, `Y = g**y mod p`, then the two parties can calculate the shared secret `ZZ = ( g**(x*y) mod p )` even though each knows only their own private key and the other party's public key. Leading zero bytes MUST be maintained in `ZZ` so it will be the same length, in bytes, as `p`. The size of `p` MUST be at least 512 bits and `g` at least 160 bits. There are numerous other complex security considerations in the selection of `g`, `p`, and a random `x` as described in [ESDH].

Diffie-Hellman key agreement is optional to implement. An example of a DH `AgreementMethod` element is as follows:

```
<AgreementMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#dh"
    ds:xmlns="http://www.w3.org/2000/09/xmldsig#">
  <KA-Nonce>Zm9v</KA-Nonce>
  <ds:DigestMethod
   Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <OriginatorKeyInfo>
    <ds:X509Data><ds:X509Certificate>
      ...
    </ds:X509Certificate></ds:X509Data>
  </OriginatorKeyInfo>
  <RecipientKeyInfo><ds:KeyValue>
    ...
  </ds:KeyValue></RecipientKeyInfo>
</AgreementMethod>
```

Assume the Diffie-Hellman shared secret is the octet sequence `ZZ`. The shared keying material needed will then be calculated as follows:

```
Keying Material = KM(1) | KM(2) | ...
```

where "|" is byte stream concatenation and

```
KM(counter) = DigestAlg ( ZZ | counter | EncryptionAlg |
              KA-Nonce | KeySize )
```

`DigestAlg`
: The message digest algorithm specified by the `DigestMethod` child of `AgreementMethod`.

`EncryptionAlg`
: The URI of the encryption algorithm, including possible key wrap algorithms, in which the derived keying material is to be used ("Example:Block/Alg" in the example above), not the URI of the agreement algorithm. This is the value of the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` grandparent of `AgreementMethod`.

`KA-Nonce`
: The base64 decoding the content of the `KA-Nonce` child of `AgreementMethod`, if present. If the `KA-Nonce` element is absent, it is null.

`Counter`
: A one byte counter starting at one and incrementing by one. It is expressed as two hex digits where letters A through F are in upper case.

`KeySize`
: The size in bits of the key to be derived from the shared secret as the UTF-8 string for the corresponding decimal integer with only digits in the string and no leading zeros. For some algorithms the key size is inherent in the URI. For others, such as most stream ciphers, it must be explicitly provided.

For example, the initial (KM(1)) calculation for the EncryptionMethod of the Key Agreement example (section 5.5) would be as follows, where the binary one byte counter value of 1 is represented by the two character UTF-8 sequence 01, ZZ is the shared secret, and "foo" is the base64 decoding of "Zm9v".

```
SHA-1 ( ZZ01Example:Block/Algfoo80 )
```

Assuming that ZZ is 0xDEADBEEF, that would be

```
SHA-1( 0xDEADBEEF30314578616D706C653A426C6F636B2F416C67666F6F3830 )
```

whose value is

```
0x534C9B8C4ABDCB50038B42015A181711068B08C1
```

Each application of DigestAlg for successive values of Counter will produce some additional number of bytes of keying material. From the concatenated string of one or more KM's, enough leading bytes are taken to meet the need for an actual key and the remainder discarded. For example, if DigestAlg is SHA-1 which produces 20 octets of hash, then for 128 bit AES the first 16 bytes from KM(1) would be taken and the remaining 4 bytes discarded. For 256 bit AES, all of KM(1) suffixed with the first 12 bytes of KM(2) would be taken and the remaining 8 bytes of KM(2) discarded.

5.6 Symmetric Key Wrap

Symmetric Key Wrap algorithms are shared secret key encryption algorithms especially specified for encrypting and decrypting symmetric keys. Their identifiers appear as Algorithm attribute values to EncryptionMethod elements that are children of EncryptedKey which is in turn a child of ds:KeyInfo which is in turn a child of EncryptedData or another EncryptedKey. The type of the key being wrapped is indicated by the Algorithm attribute of EncryptionMethod child of the parent of the ds:KeyInfo grandparent of the EncryptionMethod specifying the symmetric key wrap algorithm.

5.6.1 CMS Key Checksum

Some key wrap algorithms make use of a key checksum as defined in CMS [CMS-Wrap]. The algorithm that provides an integrity check value for the key being wrapped is:

1. Compute the 20 octet SHA-1 hash on the key being wrapped.
2. Use the first 8 octets of this hash as the checksum value.

5.6.2 CMS Triple DES Key Wrap

Identifiers and Requirements:

http://www.w3.org/2001/04/xmlenc#kw-tripledes (REQUIRED)

XML Encryption implementations MUST support TRIPLEDES wrapping of 168 bit keys and may optionally support TRIPLEDES wrapping of other keys.

An example of a TRIPLEDES Key Wrap EncryptionMethod element is as follows:

```
<EncryptionMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"/>
```

The following algorithm wraps (encrypts) a key (the wrapped key, WK) under a TRIPLEDES key-encryption-key (KEK) as adopted from [CMS-Algorithms]:

1. Represent the key being wrapped as an octet sequence. If it is a TRIPLEDES key, this is 24 octets (192 bits) with odd parity bit as the bottom bit of each octet.
2. Compute the CMS key checksum (section 5.6.1) call this CKS.
3. Let WKCKS = WK || CKS, where || is concatenation.
4. Generate 8 random octets [RANDOM] and call this IV.
5. Encrypt WKCKS in CBC mode using KEK as the key and IV as the initialization vector. Call the results TEMP1.
6. Let TEMP2 = IV || TEMP1.
7. Reverse the order of the octets in TEMP2 and call the result TEMP3.
8. Encrypt TEMP3 in CBC mode using the KEK and an initialization vector of 0x4adda22c79e82105. The resulting cipher text is the desired result. It is 40 octets long if a 168 bit key is being wrapped.

The following algorithm unwraps (decrypts) a key as adopted from [CMS-Algorithms]:

1. Check if the length of the cipher text is reasonable given the key type. It must be 40 bytes for a 168 bit key and either 32, 40, or 48 bytes for a 128, 192, or 256 bit key. If the length is not supported or inconsistent with the algorithm for which the key is intended, return error.
2. Decrypt the cipher text with TRIPLEDES in CBC mode using the `KEK` and an initialization vector (IV) of `0x4adda22c79e82105`. Call the output `TEMP3`.
3. Reverse the order of the octets in TEMP3 and call the result `TEMP2`.
4. Decompose TEMP2 into IV, the first 8 octets, and `TEMP1`, the remaining octets.
5. Decrypt TEMP1 using TRIPLEDES in CBC mode using the `KEK` and the IV found in the previous step. Call the result `WKCKS`.
6. Decompose `WKCKS`. `CKS` is the last 8 octets and `WK`, the wrapped key, are those octets before the `CKS`.
7. Calculate a CMS key checksum (section 5.6.1) over the `WK` and compare with the `CKS` extracted in the above step. If they are not equal, return error.
8. `WK` is the wrapped key, now extracted for use in data decryption.

5.6.3 AES KeyWrap

Identifiers and Requirements:

http://www.w3.org/2001/04/xmlenc#kw-aes128 (REQUIRED)
http://www.w3.org/2001/04/xmlenc#kw-aes192 (OPTIONAL)
http://www.w3.org/2001/04/xmlenc#kw-aes256 (REQUIRED)

La implementación del ajuste de claves AES se describe a continuación, según lo sugerido por NIST. Proporciona confidencialidad e integridad. Este algoritmo se define sólo para entradas que sean múltiplos de 64 bits. La información incluida no tiene por qué ser en realidad una clave. El algoritmo es el mismo independientemente del tamaño de la clave AES utilizada en el empaquetado, denominada clave de cifrado de clave o `KEK`. Los requisitos de implementación se indican a continuación.

Clave de cifrado de clave AES de 128 bits
SE REQUIERE la implementación de envoltura de claves de 128 bits.
Envoltura de otros tamaños de llaves OPCIONAL.
Clave de cifrado de clave AES de 192 bits
Todo el soporte es OPCIONAL.
Clave de cifrado de clave AES de 256 bits
SE REQUIERE la implementación de envoltura de claves de 256 bits.
Envoltura de otros tamaños de llaves OPCIONAL.

Supongamos que los datos que se van a empaquetar constan de `N`bloques de datos de 64 bits denominados `P(1)`, `P(2)`, `P(3)`... `P(N)`. El resultado del ajuste serán `N+1`bloques de 64 bits denominados `C(0)`, `C(1)`, `C(2)`, ... `C(N)`. La clave de cifrado de claves está representada por `K`. Suponga números enteros `i`, `j`y `t`un registro intermedio de 64 bits `A`, un registro de 128 bits `B`y una matriz de cantidades de 64 bits `R(1)`hasta `R(N)`.

"|" representa la concatenación, entonces `x|y`, donde `x` y `y`y cantidades de 64 bits, es la cantidad de 128 bits `x`en los bits más significativos y `y`en los bits menos significativos. `AES(K)enc(x)`es la operación de AES que cifra la cantidad de 128 bits `x`bajo la clave `K`. `AES(K)dec(x)`es la opción de descifrado correspondiente. `XOR(x,y)`es el exclusivo bit a bit o de `x`y `y`. `MSB(x)`y `LSB(y)`son los 64 bits más significativos y los 64 bits menos significativos de `x`e`y` respectivamente.

Si `N`es 1, se realiza una única operación AES para envolver o desenvolver. Si es `N>1`, entonces `6*N`se realizan operaciones AES para envolver o desenvolver.

El algoritmo de ajuste de claves es el siguiente:

1. Si `N`es `1`:
   - `B=AES(K)enc(0xA6A6A6A6A6A6A6A6|P(1))`
   - `C(0)=MSB(B)`
   - `C(1)=LSB(B)`

   Si es así `N>1`, realice los siguientes pasos:
2. Inicializar variables:
   - Establecer `A`en`0xA6A6A6A6A6A6A6A6`
   - Para , `i=1_N`
     `R(i)=P(i)`
3. Calcular valores intermedios:
   - Para , `j=0_5`
     - Para , `i=1_N`
       ```
       t= i + j*N
       B=AES(K)enc(A|R(i))
       A=XOR(t,MSB(B))
       R(i)=LSB(B)
       ```
4. Salida de los resultados:

- Colocar`C(0)=A`
- Para , `i=1_N`
  `C(i)=R(i)`

El algoritmo de desenvolvimiento de claves es el siguiente:

1. Si `N`es `1`:
   - `B=AES(K)dec(C(0)|C(1))`
   - `P(1)=LSB(B)`
   - Si `MSB(B)`es así `0xA6A6A6A6A6A6A6A6`, devuelve el éxito. De lo contrario, devolverá un error de falla de verificación de integridad.

   Si `N`>1, realice los siguientes pasos:
2. Inicialice las variables:
   - `A=C(0)`
   - Para , `i=1_N`
     `R(i)=C(i)`
3. Calcular valores intermedios:
   - Para , `j=5_0`
     - Para , `i=N_1`
       `t= i + j*N`
       `B=AES(K)dec(XOR(t,A)|R(i))`
       `A=MSB(B)`
       `R(i)=LSB(B)`
4. Salida de los resultados:
   - Para , `i=1_N`
     `P(i)=R(i)`
   - Si `A`es así `0xA6A6A6A6A6A6A6A6`, devuelve el éxito. De lo contrario, devolverá un error de falla de verificación de integridad.

Por ejemplo, envolver los datos `0x00112233445566778899AABBCCDDEEFF`con produce `KEK` `0x000102030405060708090A0B0C0D0E0F`el texto cifrado de `0x1FA68B0A8112B447`,, `0xAEF34BD8FB5A7B82`. `0x9D3E862371D2CFE5`

5.7 Resumen de mensajes

Los algoritmos de resumen de mensajes se pueden utilizar `AgreementMethod`como parte de la derivación de claves, dentro del cifrado RSA-OAEP como función hash y en conexión con el método del código de autenticación de mensajes HMAC como se describe en [XML-DSIG].)

5.7.1 SHA1

Identificador:
   http://www.w3.org/2000/09/xmldsig#sha1 (REQUERIDO)

El algoritmo SHA-1 [SHA] no toma parámetros explícitos. Un ejemplo de un `DigestMethod`elemento SHA-1 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

Un resumen SHA-1 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos. Por ejemplo, el `DigestValue`elemento para el resumen del mensaje:

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

del Apéndice A del estándar SHA-1 sería:

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

5.7.2 SHA256

Identificador:
   http://www.w3.org/2001/04/xmlenc#sha256 (RECOMENDADO)

El algoritmo SHA-256 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-256 es:

```
<DigestMethod
 Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
```

Un resumen SHA-256 es una cadena de 256 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 32 octetos.

5.7.3 SHA512

Identificador:
http://www.w3.org/2001/04/xmlenc#sha512 (OPCIONAL)

El algoritmo SHA-512 [SHA] no toma parámetros explícitos. `DigestMethod`Un ejemplo de un elemento SHA-512 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#sha512"/>
```

Un resumen SHA-512 es una cadena de 512 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 64 octetos.

5.7.4 RIPEMD-160

Identificador:
http://www.w3.org/2001/04/xmlenc#ripemd160 (OPCIONAL)

El algoritmo RIPEMD-160 [RIPEMD-160] no toma parámetros explícitos. Un ejemplo de un `DigestMethod` elemento RIPEMD-160 es:

```
<Método de resumen
 Algoritmo="http://www.w3.org/2001/04/xmlenc#ripemd160"/>
```

Un resumen RIPEMD-160 es una cadena de 160 bits. El contenido del `DigestValue`elemento será la codificación base64 de esta cadena de bits vista como un tren de octetos de 20 octetos.

5.8 Autenticación de mensajes

Identificador:
http://www.w3.org/2000/09/xmldsig# (RECOMENDADO)

La firma XML [XML-DSIG] es OPCIONAL para implementar en aplicaciones de cifrado XML. Es la forma recomendada de proporcionar autenticación basada en claves.

5.9 Canonicalización

Una canonicalización de XML es un método para serializar XML de forma coherente en un flujo de octetos, según sea necesario antes de cifrar XML.

5.9.1 Canonicalización inclusiva

Identificadores:
http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (OPCIONAL)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (OPCIONAL)

XML canónico [Canon] es un método de serialización de XML que incluye el espacio de nombres dentro del alcance y el contexto del atributo del espacio de nombres xml de los antepasados del XML que se está serializando.

Si XML se va a cifrar y luego descifrar en un entorno diferente y se desea preservar los enlaces de prefijo del espacio de nombres y el valor de los atributos en el espacio de nombres "xml" de su entorno original, entonces se debe utilizar la versión canónica XML con comentarios del XML. Sea la serialización que esté cifrada.

5.9.2 Canonicalización exclusiva

Identificadores:
http://www.w3.org/2001/10/xml-exc-c14n# (OPCIONAL)
http://www.w3.org/2001/10/xml-exc-c14n#WithComments (OPCIONAL)

Canonicalización XML exclusiva [Exclusivo] serializa XML de tal manera que incluye en la medida mínima práctica el enlace de prefijo de espacio de nombres y el contexto de atributo de espacio de nombres xml heredado de elementos ancestros.

Es el método recomendado donde se puede cambiar el contexto externo de un fragmento que fue firmado y luego cifrado. De lo contrario, la validación de la firma sobre el fragmento puede fallar porque la canonicalización mediante validación de firma puede incluir espacios de nombres innecesarios en el fragmento.

6 consideraciones de seguridad

6.1 Relación con las firmas digitales XML

La aplicación de cifrado y firmas digitales en partes de un documento XML puede dificultar el descifrado y la verificación de la firma posteriores. En particular, al verificar una firma se debe saber si la firma se calculó sobre la forma de elementos cifrados o no cifrados.

Un problema aparte, pero importante, es la introducción de vulnerabilidades criptográficas al combinar firmas digitales y cifrado sobre un elemento XML común. Hal Finney ha sugerido que cifrar datos firmados digitalmente, dejando la firma digital en claro, puede permitir ataques de adivinación de texto sin formato. Esta vulnerabilidad se puede mitigar mediante el uso de hashes seguros y nonces en el texto que se procesa.

De acuerdo con el documento de requisitos [EncReq], la interacción entre cifrado y firma es un problema de aplicación y está fuera del alcance de la especificación. Sin embargo, hacemos las siguientes recomendaciones:

1. Cuando los datos están cifrados, cualquier resumen o firma sobre esos datos debe cifrarse. Esto satisface el primer problema en el sentido de que sólo se pueden validar aquellas firmas que se pueden ver. También aborda la posibilidad de una vulnerabilidad de adivinación de texto sin formato, aunque puede que no sea posible identificar (o incluso conocer) todas las firmas de un determinado dato.
2. Emplee la transformación de firma "decrypt-except" [XML-DSIG-Decrypt] . Funciona de la siguiente manera: durante el procesamiento de transformación de firma, si encuentra una transformación de descifrado, descifre todo el contenido cifrado del documento excepto aquellos exceptuados por un conjunto enumerado de referencias.

Además, si bien las siguientes advertencias se refieren a inferencias incorrectas por parte del usuario sobre la autenticidad de la información cifrada, las aplicaciones deben disuadir la mala interpretación del usuario comunicando claramente qué información tiene integridad o está autenticada, es confidencial o no repudiable cuando se realizan múltiples procesos (por ejemplo, firma). y cifrado) y se utilizan algoritmos (por ejemplo, simétricos y asimétricos):

1. Cuando un sobre cifrado contiene una firma, la firma no necesariamente protege la autenticidad o integridad del texto cifrado [Davis].
2. Si bien la firma protege el texto sin formato, solo cubre lo que está firmado, los destinatarios de los mensajes cifrados no deben inferir la integridad o autenticidad de otra información sin firmar (por ejemplo, encabezados) dentro del sobre cifrado; consulte [XML-DSIG , 8.1.1 Solo lo que está firmado es seguro].

6.2 Información revelada

Cuando una clave simétrica se comparte entre varios destinatarios, esa clave simétrica *solo* debe usarse para los datos destinados a *todos* los destinatarios; Incluso si un destinatario no es dirigido a información destinada (exclusivamente) a otro en la misma clave simétrica, la información podría ser descubierta y descifrada.

Además, los diseñadores de aplicaciones deben tener cuidado de no revelar ninguna información en los parámetros o identificadores de algoritmos (por ejemplo, información en un URI) que debilite el cifrado.

6.3 Nonce y IV (Valor o Vector de Inicialización)

Una característica indeseable de muchos algoritmos de cifrado y/o sus modos es que el mismo texto sin formato, cuando se cifra con la misma clave, tiene el mismo texto cifrado resultante. Si bien esto no es sorprendente, invita a varios ataques que se mitigan al incluir datos arbitrarios y no repetidos (bajo una clave determinada) con el texto sin formato antes del cifrado. En los modos de encadenamiento de cifrado, estos datos son los primeros en cifrarse y, en consecuencia, se denominan IV (valor de inicialización o vector).

Los diferentes algoritmos y modos tienen requisitos adicionales sobre las características de esta información (por ejemplo, aleatoriedad y secreto) que afectan las características (por ejemplo, confidencialidad e integridad) y su resistencia a los ataques.

Dado que los datos XML son redundantes (por ejemplo, codificaciones Unicode y etiquetas repetidas) y que los atacantes pueden conocer la estructura de los datos (por ejemplo, DTD y esquemas), los algoritmos de cifrado

deben implementarse y utilizarse cuidadosamente en este sentido.

Para el modo Cipher Block Chaining (CBC) utilizado por esta especificación, el IV no debe reutilizarse para ninguna clave y debe ser aleatorio, pero no es necesario que sea secreto. Además, en este modo, un adversario que modifique el IV puede realizar un cambio conocido en el texto sin formato después del descifrado. Este ataque se puede evitar asegurando la integridad de los datos de texto sin formato, por ejemplo firmándolos.

6.4 Denegación de Servicio

Esta especificación permite el procesamiento recursivo. Por ejemplo, es posible el siguiente escenario: `EncryptedKey` **A** requiere que `EncryptedKey` **B** sea descifrado, lo que a su vez requiere `EncryptedKey` **A** ! O bien, un atacante podría enviar un mensaje `EncryptedData`de descifrado que haga referencia a recursos de red que son muy grandes o que se redirigen continuamente. En consecuencia, las implementaciones deberían poder restringir la recursividad arbitraria y la cantidad total de recursos de red y procesamiento que una solicitud puede consumir.

6.5 Contenido inseguro

El cifrado XML se puede utilizar para ocultar, mediante cifrado, contenido que las aplicaciones (por ejemplo, cortafuegos, detectores de virus, etc.) consideran inseguro (por ejemplo, código ejecutable, virus, etc.). En consecuencia, dichas aplicaciones deben considerar que el contenido cifrado es tan inseguro como el contenido más inseguro transportado en el contexto de su aplicación. En consecuencia, dichas aplicaciones pueden optar por (1) no permitir dicho contenido, (2) exigir acceso al formulario descifrado para su inspección o (3) garantizar que el contenido arbitrario pueda procesarse de forma segura al recibir las aplicaciones.

7 Conformidad

Una implementación cumple con esta especificación si genera con éxito la sintaxis de acuerdo con las definiciones del esquema y satisface todos los requisitos MUST/REQUIRED/SHALL, incluido el soporte y el procesamiento de algoritmos . Los requisitos de procesamiento se especifican sobre las funciones de ***descifrador* , *cifrador* y** su ***aplicación*** de llamada .

8 Tipo de medio de cifrado XML

8.1 Introducción

Sintaxis y procesamiento de cifrado XML [XML-Encryption] especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden ser datos arbitrarios (incluido un documento XML), un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

El `application/xenc+xml`tipo de medio permite que las aplicaciones de cifrado XML identifiquen documentos cifrados. Además, permite que las aplicaciones que conocen este tipo de medio (incluso si no son implementaciones de cifrado XML) tengan en cuenta que el tipo de medio del objeto descifrado (original) puede ser un tipo distinto de XML.

8.2 aplicación/xenc+xml Registro

Este es un registro de tipo de medio tal como se define en Extensiones multipropósito de correo de Internet (MIME), parte cuatro: Procedimientos de registro [MIME-REG]

Nombre del tipo de medio MIME: aplicación

Nombre del subtipo MIME: xenc+xml

Parámetros requeridos: ninguno

Parámetros opcionales: juego de caracteres

> Los valores permitidos y recomendados y la interpretación del parámetro charset son idénticos a los proporcionados para 'application/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de codificación:

> Las consideraciones de codificación son idénticas a las dadas para 'aplicación/xml' en la sección 3.2 de RFC 3023 [XML-MT].

Consideraciones de Seguridad:

Consulte la sección Consideraciones de seguridad [Cifrado XML] .

Consideraciones de interoperabilidad: ninguna

Especificación publicada: [Cifrado XML]

Aplicaciones que utilizan este tipo de medio:

XML Encryption es neutral en cuanto a dispositivos, plataformas y proveedores y es compatible con una variedad de aplicaciones web.

Información adicional:

Número(s) mágico(s): ninguno

Aunque no se puede contar con secuencias de bytes para identificar consistentemente los documentos XML Encryption, serán documentos XML en los que el elemento raíz ' `QName`s `LocalPart`es `'EncryptedData'`o ' `EncryptedKey`' con un nombre de espacio de nombres asociado de ' http://www.w3.org/2001/ 04/xmlenc# '. El `application/xenc+xml`nombre del tipo DEBE usarse solo para objetos de datos en los que el elemento raíz sea del espacio de nombres de cifrado XML. Los documentos XML que contienen estos tipos de elementos en lugares distintos del elemento raíz se pueden describir utilizando funciones como [esquema XML].

Extensiones de archivo: .xml

Código(s) de tipo de archivo de Macintosh: "TEXTO"

Persona y dirección de correo electrónico a contactar para obtener más información:

José Reagle <reagle@w3.org>

Grupo de trabajo XENC <xml-encryption@w3.org>

Uso previsto: COMÚN

Autor/Cambiar controlador:

La especificación de cifrado XML es un producto del trabajo del World Wide Web Consortium (W3C), que tiene control de cambios sobre la especificación.

9 esquema y ejemplos válidos

Esquema
esquema-xenc.xsd
Ejemplo
enc-example.xml (no es criptográficamente válido pero ejercita gran parte del esquema)

10 referencias

TRIPLES
ANSI X9.52: Modos de operación del algoritmo de cifrado de datos triple. 1998.
AES
NIST FIPS 197: Estándar de cifrado avanzado (AES) . Noviembre de 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
ENVOLTURA AES
RFC3394: Algoritmo de ajuste de claves del estándar de cifrado avanzado (AES) . J. Schaad y R. Housley. Informativo, septiembre de 2002.
Algoritmos CMS
RFC3370: Algoritmos de sintaxis de mensajes criptográficos (CMS) . R. Housley. Informativo, febrero de 2002.
http://www.ietf.org/rfc/rfc3370.txt
Envoltura CMS
RFC3217: Encapsulación de claves Triple-DES y RC2 . R. Housley. Informativo, diciembre de 2001.
http://www.ietf.org/rfc/rfc3217.txt
davis
Firma y cifrado defectuosos en S/MIME, PKCS#7, MOSS, PEM, PGP y XML. D. Davis. Conferencia Técnica Anual de USENIX. 2001.
http://www.usenix.org/publications/library/proceedings/usenix01/davis.html

DES
NIST FIPS 46-3: Estándar de cifrado de datos (DES). Octubre de 1999.
http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
EncReq
Requisitos de cifrado XML . J. Reagle. Nota del W3C, marzo de 2002.
http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304
ESDH
RFC 2631: Método de acuerdo de claves Diffie-Hellman. E. Rescorla. Seguimiento de estándares, 1999.
http://www.ietf.org/rfc/rfc2631.txt
http://www.w3.org/TR/2002/CR-xml-exc-c14n-20020212
Glosario
RFC 2828: Glosario de seguridad de Internet . R Shirey. Informativo, mayo de 2000.
http://www.ietf.org/rfc/rfc2828.txt
HMAC
RFC 2104: HMAC: hash con clave para autenticación de mensajes . H. Krawczyk, M. Bellare y R. Canetti. Informativo, febrero de 1997.
http://www.ietf.org/rfc/rfc2104.txt
HTTP
RFC 2616: Protocolo de transferencia de hipertexto - HTTP/1.1. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee. Standards Track, junio de 1999.
http://www.ietf.org/rfc/rfc2616.txt
PALABRAS CLAVE
RFC 2119: Palabras clave para uso en RFC para indicar niveles de requisitos. S. Bradner. Mejores prácticas actuales, marzo de 1997.
http://www.ietf.org/rfc/rfc2119.txt
MD5
RFC 1321: Algoritmo de resumen de mensajes MD5. R. Rivest. Informativo, abril de 1992.
http://www.ietf.org/rfc/rfc1321.txt
MÍMICA
RFC 2045: Extensiones multipropósito de correo de Internet (MIME), primera parte: Formato de los cuerpos de los mensajes de Internet . N. Freed y N. Borenstein. Standards Track, noviembre de 1996.
http://www.ietf.org/rfc/rfc2045.txt
MIME-REG
RFC 2048: Extensiones multipropósito de correo de Internet (MIME), cuarta parte: Procedimientos de registro . N. Freed, J. Klensin y J. Postel. Mejores prácticas actuales, noviembre de 1996.
http://www.ietf.org/rfc/rfc2048.txt
NFC
TR15, formularios de normalización Unicode . M. Davis y M. Dürst. Revisión 18: noviembre de 1999.
http://www.unicode.org/unicode/reports/tr15/tr15-18.html .
Corrección NFC
"Corrigéndum n.º 2: Yod con normalización de Hiriq ".
http://www.unicode.org/versions/corrigendum2.html .
prop1
"Propuesta de hombre de paja de cifrado XML ". E. Simon y B. LaMacchia. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0001.html
prop2
Otra propuesta de Cifrado XML . T. Imamura. Agosto de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Aug/0005.html
prop3
Sintaxis y procesamiento de cifrado XML . B. Dillaway, B. Fox, T. Imamura, B. LaMacchia, H. Maruyama, J. Schaad y E. Simon. Diciembre de 2000.
http://lists.w3.org/Archives/Public/xml-encryption/2000Dec/att-0024/01-XMLEncryption_v01.html
PKCS1
RFC 2437: PKCS #1: Especificaciones de criptografía RSA Versión 2.0. B. Kaliski y J. Staddon. Informativo, octubre de 1998.
http://www.ietf.org/rfc/rfc2437.txt
ALEATORIO
RFC 1750: Recomendaciones de aleatoriedad para la seguridad . D. Eastlake, S. Crocker y J. Schiller. Informativo, diciembre de 1994.
http://www.ietf.org/rfc/rfc1750.txt
RIPEMD-160
CryptoBytes, Volumen 3, Número 2. La función hash criptográfica RIPEMD-160 . Laboratorios RSA. Otoño de 1997.
ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf
http://www.esat.kuleuven.ac.be/~cosicart/pdf/AB-9601/AB-9601.pdf
sha
Estándar de hash seguro . NIST FIPS 180-1. (RFC 3174). Abril de 1995.
http://www.itl.nist.gov/fipspubs/fip180-1.htm

Estándar de hash seguro. Borrador NIST FIPS 180-2 . 2001. (Ampliado para incluir SHA-384, SHA-256 y SHA-512)
http://csrc.nist.gov/encryption/shs/dfips-180-2.pdf

A Bin
R. Tobin. Conjunto de información para entidades externas , lista de correo XML Core, 2000 [solo miembro del W3C].
http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000OctDec/0054

UTF-16
RFC 2781: UTF-16, una codificación de ISO 10646. P. Hoffman y F. Yergeau. Informativo, febrero de 2000.
http://www.ietf.org/rfc/rfc2781.txt

UTF-8
RFC 2279: UTF-8, un formato de transformación de ISO 10646F. F. Yergeau. Standards Track, enero de 1998.
http://www.ietf.org/rfc/rfc2279.txt

URI
RFC 2396: Identificadores uniformes de recursos (URI): sintaxis genérica . T. Berners-Lee, R. Fielding y L. Masinter. Standards Track, agosto de 1998.
http://www.ietf.org/rfc/rfc2396.txt
http://www.ietf.org/rfc/rfc1738.txt
http://www.ietf.org/rfc/rfc2141.txt
RFC 2611: Mecanismos de definición de espacios de nombres URN. Mejores prácticas actuales. Daigle, D. van Gulik, R. Iannella, P. Falstrom. Junio de 1999.
http://www.ietf.org/rfc/rfc2611.txt

X509v3
Recomendación UIT-T X.509 versión 3 (1997). "Tecnología de la información - Interconexión de sistemas abiertos - El marco de autenticación de directorios" ISO/IEC 9594-8:1997.

XML
Lenguaje de marcado extensible (XML) 1.0 (segunda edición) . T. Bray, J. Paoli, CM Sperberg-McQueen y E. Maler. Recomendación del W3C, octubre de 2000.

Base XML
Base XML . J. Marsh. Recomendación del W3C, junio de 2001.
http://www.w3.org/TR/2001/REC-xmlbase-20010627/

XML-C14N
XML canónico. J. Boyer. Recomendación del W3C, marzo de 2001.
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.ietf.org/rfc/rfc3076.txt

XML-exc-C14N
Canonicalización XML exclusiva . J. Boyer, D. Eastlake y J. Reagle. Recomendación del W3C, julio de 2002.
http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/

XML-DSIG
Sintaxis y procesamiento de firmas XML . D. Eastlake, J. Reagle y D. Solo. Recomendación del W3C, febrero de 2002.
http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

XML-DSIG-Descifrar
Transformación de descifrado para firma XML . M. Hughes, T. Imamura y H. Maruyama. Recomendación del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210

Cifrado XML
Sintaxis y procesamiento de cifrado XML . D. Eastlake y J. Reagle. Recomendación candidata del W3C, diciembre de 2002.
http://www.w3.org/TR/2002/CR-xmlenc-core-20020802/

Conjunto de información XML
Conjunto de información XML . J. Cowan y R. Tobin. Recomendación del W3C, octubre de 2001
http://www.w3.org/TR/2001/REC-xml-infoset-20011024/

XML-MT
RFC 3023: tipos de medios XML. M. Murata, S. St. Laurent y D. Kohn. Informativo, enero de 2001.
http://www.ietf.org/rfc/rfc2376.txt

XML-NS
Espacios de nombres en XML . T. Bray, D. Hollander y A. Layman. Recomendación del W3C, enero de 1999.
http://www.w3.org/TR/1999/REC-xml-names-19990114

esquema XML
Esquema XML Parte 1: Estructuras D. Beech, M. Maloney y N. Mendelsohn. Recomendación del W3C, mayo de 2001.
http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
Esquema XML, parte 2: tipos de datos . P. Biron y A. Malhotra. Recomendación del W3C, mayo de 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

XPath
Lenguaje de ruta XML (XPath) versión 1.0 . J. Clark y S. DeRose. Recomendación del W3C, octubre de 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116